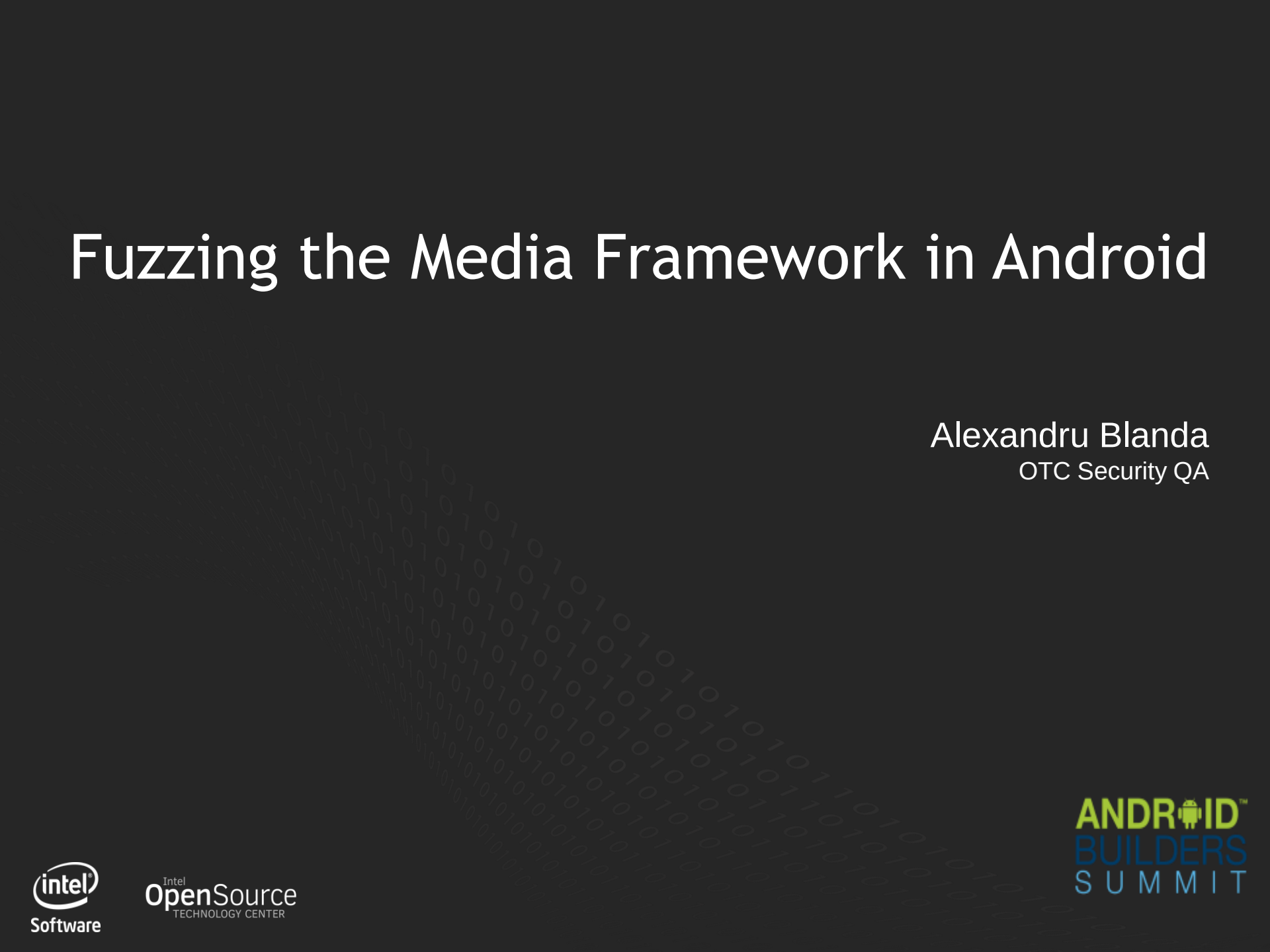
Fuzzing the Media Framework in Android

Alexandru Blanda
OTC Security QA



intel Software
Intel OpenSource TECHNOLOGY CENTER
ANDROID BUILDERS SUMMIT

ANDROID BUILDERS SUMMIT

Agenda

Introduction

Fuzzing Media Content in Android

Data Generation

Fuzzing the Stagefright Framework

Logging & Triage Mechanisms

Introduction

Fuzzing

- Form of black-box testing

- Involves sending corrupt input to a software system and monitoring for crashes

- Purpose: find security-related problems or any other critical defects that could lead to an undesirable behaviour of the system

Introduction

Fuzzing

Possible targets:

- Media Players
- Document Viewers
- Web Browsers
- Antivirus products
- Binary (ELF)

Introduction

Audio and video as attack vectors

- Binary streams containing complex data
- Large variety of audio and video players and associated media codecs
- User perception that media files are harmless
- Media playback doesn't require special permissions

Introduction

What to expect

- Crashes (SIGSEGV, SIGFPE, SIGABRT, SIGILL)

- Process hangs (synchronization issues, memory leaks, infinite loops)

- Denial of Service situations (device reboots, application crashes)

- Buffer overflows, null-pointer dereference, integer overflows

ANDROID BUILDERS SUMMIT

Agenda

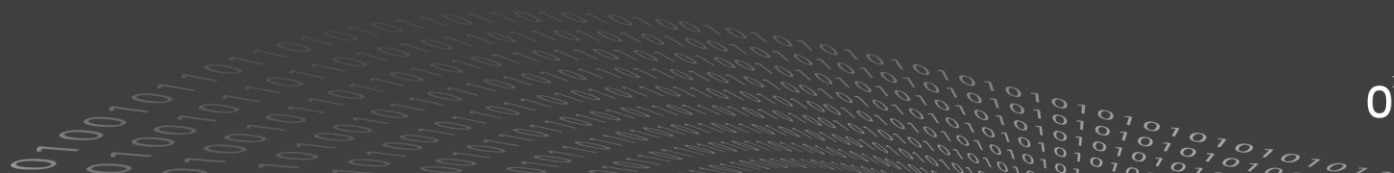
Introduction

Fuzzing Media Content in Android

Data Generation

Fuzzing the Stagefright Framework

Logging & Triage Mechanisms



intel Software

Intel OpenSource TECHNOLOGY CENTER

Fuzzing Media Content in Android

ANDROID BUILDERS SUMMIT

Overview

- Create corrupt but structurally valid media files
- Direct them to the appropriate decoders in Android
- Monitor the system for potential issues
- Pass the issues through a triage mechanism

Fuzzing Media Content in Android

ANDROID BUILDERS SUMMIT

Steps in a fuzzing campaign

1. Identify type of input
2. Identify entry point in the system
3. Data generation
4. Execution phase (actual fuzzing process)
5. Monitor results (logging process)
6. Triage phase

Fuzzing Media Content in Android

ANDROID BUILDERS SUMMIT

Steps in a fuzzing campaign

1. Identify type of input - corrupt media files
2. Identify entry point in the system - Stagefright framework
3. Data generation - various fuzzing tools
4. Execution phase - Stagefright CLI
5. Monitor results - log buffer in Android
6. Triage phase - /data/tombstones

ANDROID BUILDERS SUMMIT

Agenda

Introduction

Fuzzing Media Content in Android

Data generation

Fuzzing the Stagefright framework

Logging & Triage mechanisms

Data generation

Tools

- Basic Fuzzing Framework (BFF)
- FuzzBox
- Radamsa
- American Fuzzy Lop (AFL)
- Seed gathering

Data generation

Basic Fuzzing Framework (BFF)

- Mutational fuzzing on software that consumes file input
- Automatically generated GDB and Valgrind traces
- Crash classification based on bug severity/exploitability degree
- Automated test case minimization, for inputs that produce a crash
- Based on a modified version of zzuf

Data generation

BFF for Android fuzzing

ANDROID BUILDERS SUMMIT

- Generate test files on a temporary location the disk (rather than directly in memory)
- External script to save the files from the temporary location
- Retrace generated test cases to their initial seed files

Data generation

FuzzBox

- Multi-codec media fuzzing tool, written in Python
- Creates corrupt but structurally valid media files and launches them in a player, while gathering GDB backtraces
- More targeted than BFF (targets specific stream formats)
- Supported filetypes: Ogg, FLAC, ASF(WMV, WMA), MP3, MP4, Speex, WAV, AIFF

Data generation

FuzzBox for Android fuzzing

- Several changes from the standard tool:
  - Only use the data generation functionality of the tool
  - Retrace all generated test files to their initial seed files
  - Automated tool usage

- Much faster than BFF !

Data generation

Radamsa

- General purpose fuzzer
- Random, deterministic, model-based fuzzer
- Collection of ~15 smaller model-based fuzzers
- Control over mutation patterns and data generation sources
- Mainly used only for generating test cases
- Can be easily ported to run directly on Android (advantages?)

Data generation

Seed gathering

- Python mass downloader using Google and Bing search engines

- The LibAv samples collection: more than 50 GB of valid and corrupt media files
http://samples.mplayerhq.hu/

- -inurl:htm -inurl:html intitle: ”index of” .mp3 + wget

```
./google-downloader.py --help
Usage: google-downloader.py [options]

Options:
  -h, --help            show this help message and exit
  -s SEARCH, --search=SEARCH
                        keyword to SEARCH
  -n NUM, --number=NUM  Number of results to SEARCH
  -d DOMAIN, --domain=DOMAIN
                        The url you want google.com or
google.co.in, all you
                        have to do is enter 'com' or
'co.in' etc.
  -l LANGUAGE, --language=LANGUAGE
                        Select your language (Default en)
```

ANDROID BUILDERS SUMMIT

Agenda

Introduction

Fuzzing Media Content in Android

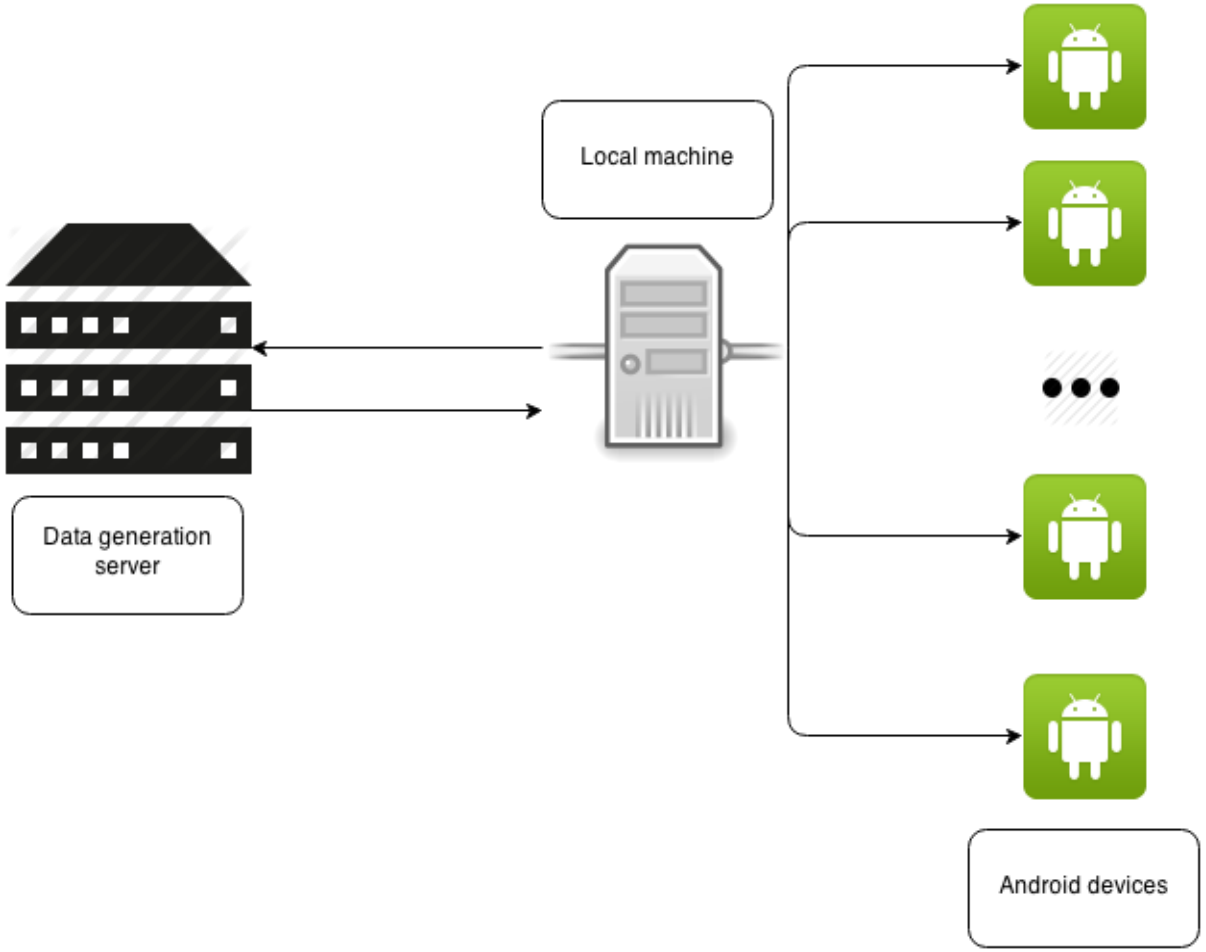
Data generation

Fuzzing the Stagefright framework

Logging & Triage mechanisms

Fuzzing the Stagefright framework

The fuzzing infrastructure



Fuzzing the Stagefright framework

Overview of the testing process

- Corrupted media input is created on a server using the data generation tools
- The server sends large sets of test cases to the local host
- Each set of test files is automatically divided into separate batches
- Each device receives a batch of testing files in a distributed manner and logs the results separately

Fuzzing the Stagefright framework

ANDROID BUILDERS SUMMIT

Stagefright command line interface

```
root@android:/ # stagefright -h
usage: stagefright
-h(elp)
-a(udio)
-n repetitions
-l(ist) components
-m max-number-of-frames-to-decode in each pass
-p(rofiles) dump decoder profiles supported
-t(humbnail) extract video thumbnail or album art
-s(oftware) prefer software codec
-r(hardware) force to use hardware codec
-o playback audio
-w(rite) filename (write to .mp4 file)
-x display a histogram of decoding times/fps (video only)
-S allocate buffers from a surface
-T allocate buffers from a surface texture
-d(ump) filename (raw stream data to a file)
-D(ump) filename (decoded PCM data to a file)
```

ANDROID BUILDERS SUMMIT

Agenda

Introduction

Fuzzing Media Content in Android

Data generation

Fuzzing the Stagefright framework

Logging & Triage mechanisms

Logging and Triage Mechanisms

Logging process

- Log every test case executed with Fatal priority

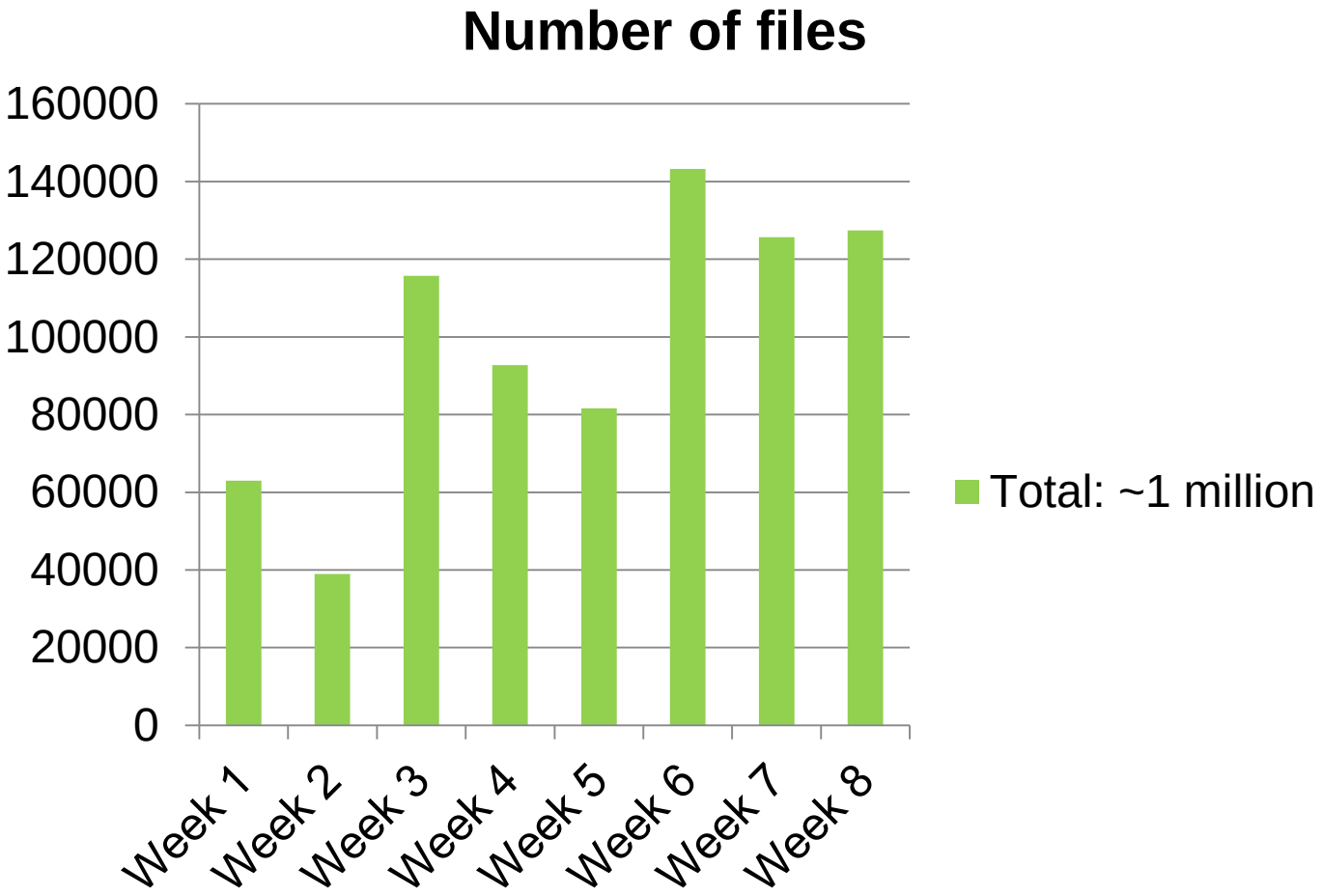
```
"adb shell log -p F -t sp_stagefright *** Filename:" + test_files[i]
```

- Save filtered logcat buffer for each campaign, for all devices

```
01-13 04:19:15.462 F/Stagefright(29791): - sp_stagefright *** 610 - Filename:zzuf.27425.SzNP6l.mkv
01-13 04:19:18.466 F/Stagefright(29822): - sp_stagefright *** 611 - Filename:zzuf.18320.vKu92z.wmv
01-13 04:20:05.150 F/Stagefright(29844): - sp_stagefright *** 612 - Filename:zzuf.2948.ciFQUs.mp4
01-13 04:20:23.182 F/Stagefright(29859): - sp_stagefright *** 613 - Filename:zzuf.30915.z1C5XH.mov
01-13 04:20:54.285 F/Stagefright(29882): - sp_stagefright *** 614 - Filename:zzuf.1607.BkHjHj.mpg
01-13 04:20:55.010 F/Stagefright(29897): - sp_stagefright *** 615 - Filename:zzuf.29755.v3EmT1.asf
01-13 04:21:10.134 F/libc    (29902): Fatal signal 11 (SIGSEGV) at 0x56579489 (code=1), thread 29902
01-13 04:21:13.769 F/Stagefright(29912): - sp_stagefright *** 616 - Filename:zzuf.19996.iXjx7V.avi
01-13 04:21:18.585 F/Stagefright(29934): - sp_stagefright *** 617 - Filename:zzuf.14298.DAOJ0a.mts
01-13 04:21:20.505 F/Stagefright(29949): - sp_stagefright *** 618 - Filename:zzuf.12202.Cmg6mz.wmv
01-13 04:21:23.165 F/Stagefright(29964): - sp_stagefright *** 619 - Filename:zzuf.2400.yA7uCg.wmv
```

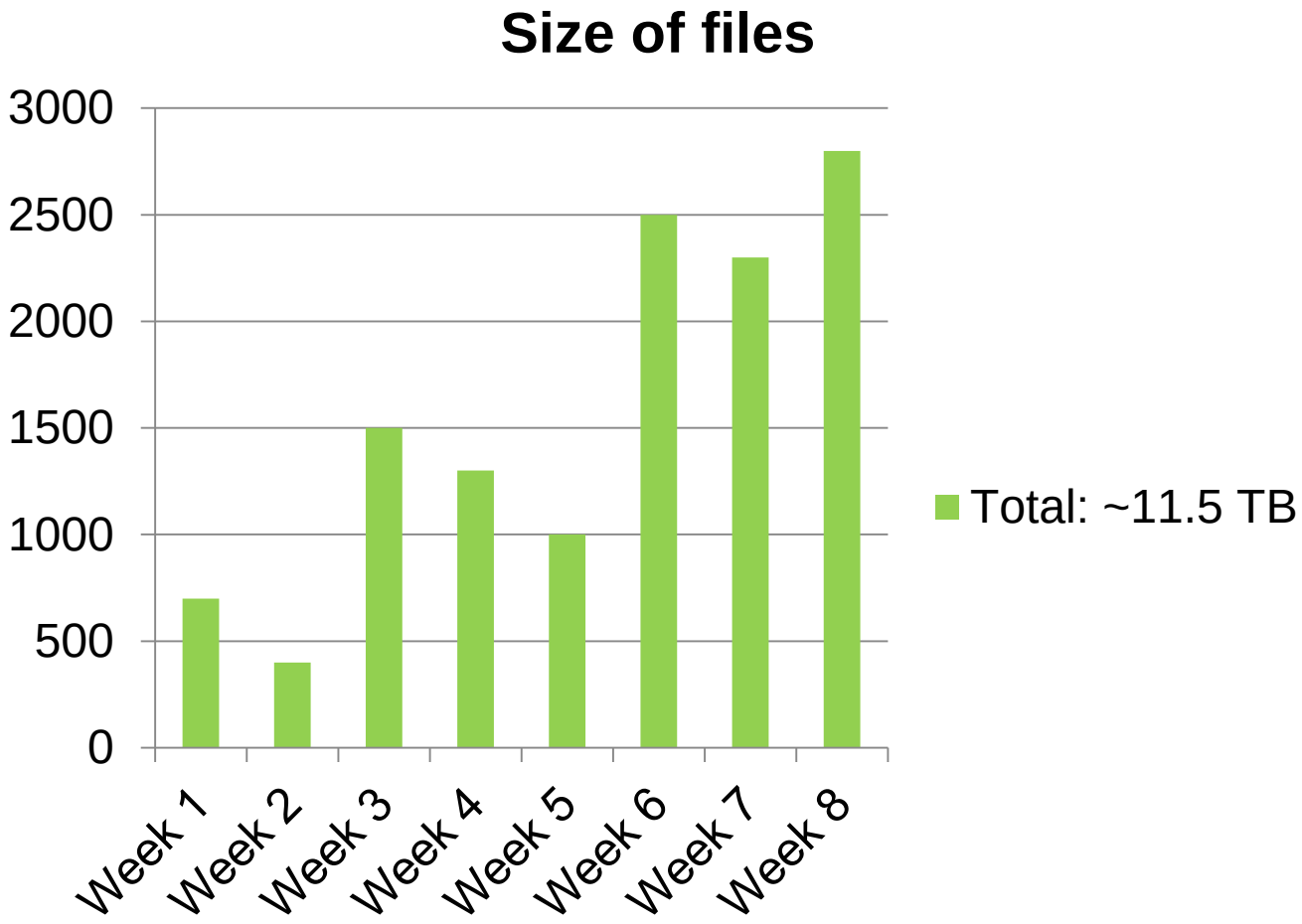
Logging and Triage Mechanisms

Initial results



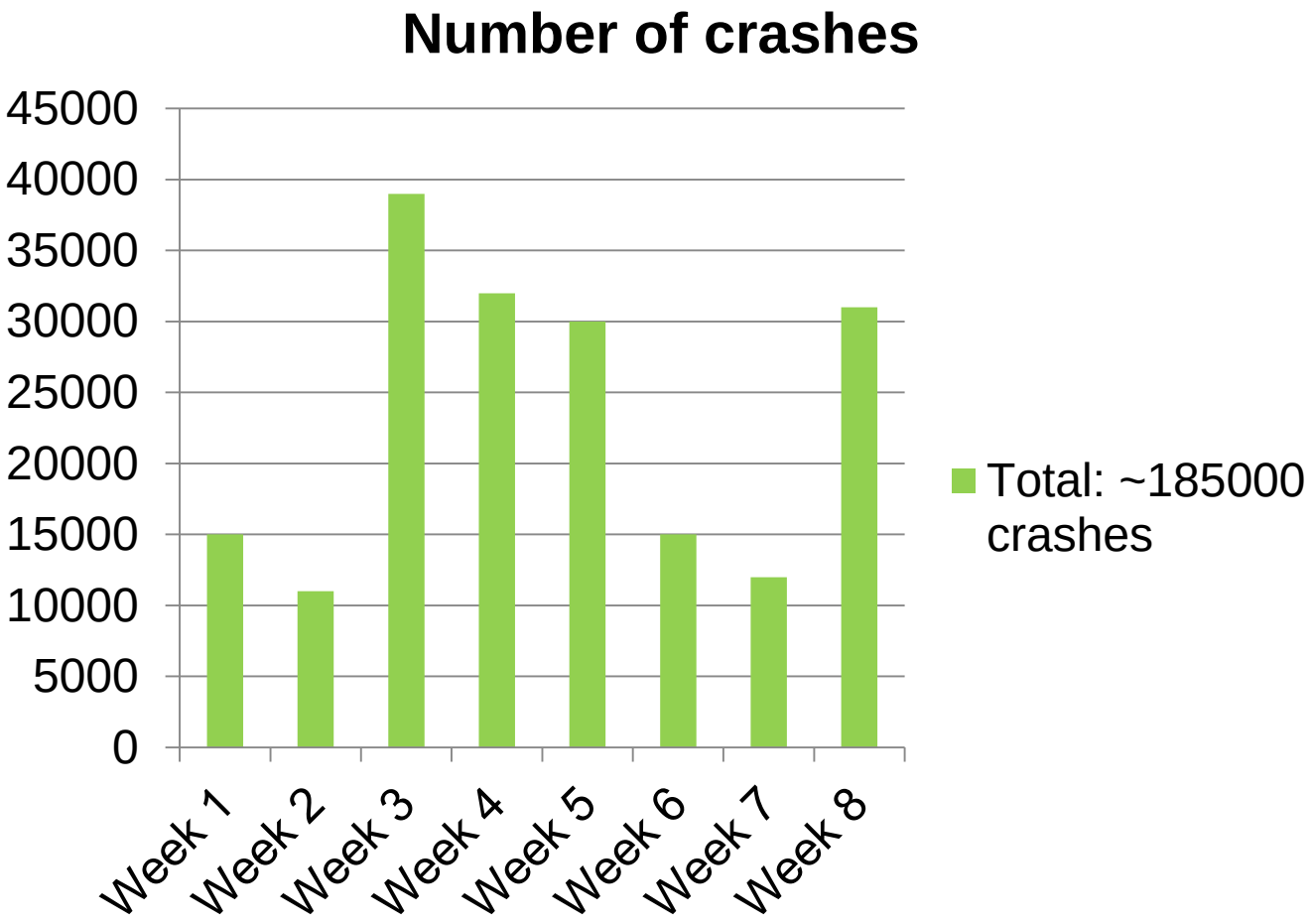
Logging and Triage Mechanisms

Initial results



Logging and Triage Mechanisms

Initial results



Logging and Triage Mechanisms

Triage phase

ANDROID BUILDERS SUMMIT

- Problem: Automated fuzzing campaigns generating large number of crashes (issues)
  - Manual sorting is not an option

- Suitable testing scenarios: involve executing various test cases on devices and monitoring for crashes

Logging and Triage Mechanisms

Testing scenario

ANDROID BUILDERS SUMMIT

2 separate phases:

- First run testing phase
  - Test cases are executed on the device
  - Logs are created during each test run
- Triage phase
  - Generated logs are parsed to identify crashing test cases
  - Crashing test cases are resent to the device
  - Previously unseen crashes get stored in the unique issues pool

Logging and Triage Mechanisms

Triage phase - implementation

- Each test case that produces a crash generates an entry in *data/tombstones* and *data/system/dropbox*

```
signal 11 (SIGSEGV), code 1 (SEGV_MAPERR), fault addr 00000000
Abort message: 'invalid address or address of corrupt block 0xf9073db8 passed to dlfree'
    eax f9053000  ebx f722ff30  ecx 00000000  edx 39077000
    esi 39075000  edi 39074e28
    xcs 00000023  xds 0000002b  xes 0000002b  xfs 00000000  xss 0000002b
    eip f718a048  ebp f9075000  esp ffb88600  flags 00010246

backtrace:
    #00  pc 00012048  /system/lib/libc.so (sys_alloc.constprop.14+1224)
    #01  pc 00012dbe  /system/lib/libc.so (dlmalloc+1022)
    #02  pc 0000cfee  /system/lib/libc.so (malloc+30)
    #03  pc 000e2b5a  /system/lib/libstagefright.so (android::MediaBuffer::MediaBuffer(unsigned int)+74)
    #04  pc 001c24cb  /system/lib/libstagefright.so (android::MediaBufferPool::acquire_buffer(int,
                      android::MediaBuffer**)+267)
    #05  pc 001c136a  /system/lib/libstagefright.so (android::AsfExtractor::readPacket()+634)
    #06  pc 001c1cf2  /system/lib/libstagefright.so (android::ASFSource::read(android::MediaBuffer**,
                      android::MediaSource::ReadOptions const*)+194)
    #07  pc 00331f76  /system/lib/libstagefright.so (android::UMCAudioDecoder<UMC::CreateWMADecoder()>::
                      read(android::MediaBuffer**, android::MediaSource::ReadOptions const*)+3046)
```

Logging and Triage Mechanisms

Triage phase - implementation

1. Parse the logs and identify the test cases that caused a crash
2. Resend the files to the testing infrastructure
3. For each test file sent:
    a. Grab the generated tombstone
    b. Parse the tombstone and get the PC value
    c. Check if the PC value has been previously encountered
    d. Save the tombstone and the test case if the issue is new

Logging and Triage Mechanisms

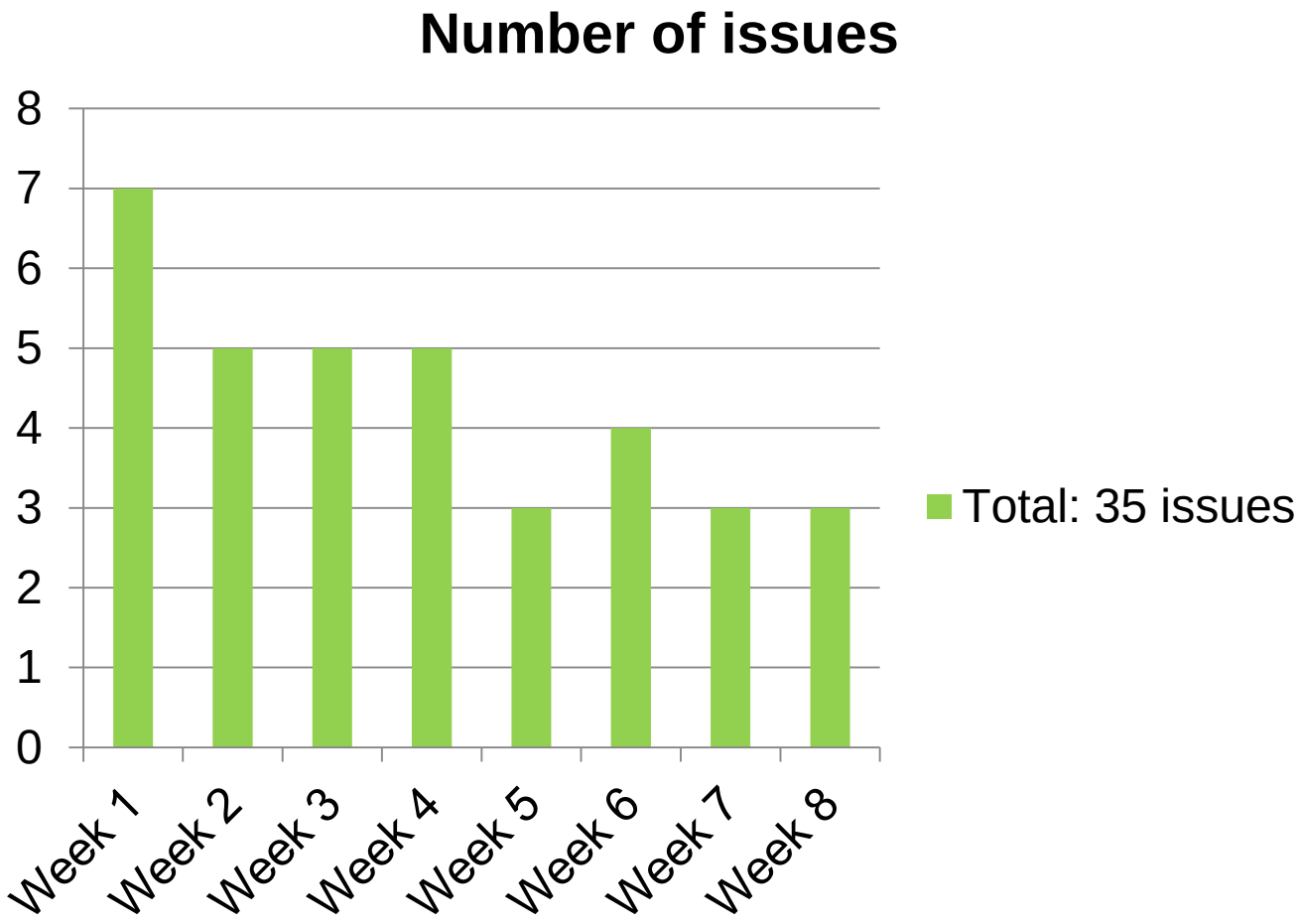
Triage phase - implementation

- Diff between the folder that contains the unique issues, before and after the triage process:

```
Common subdirectories: ./0015ae9f and old_issues/0015ae9f
Common subdirectories: ./00163774 and old_issues/00163774
Only in .: 001639cf
Only in .: 00167d90
Common subdirectories: ./00168304 and old_issues/00168304
Common subdirectories: ./00169d0f and old_issues/00169d0f
Common subdirectories: ./0016c8a7 and old_issues/0016c8a7
Only in .: 001a9211
Common subdirectories: ./00235a99 and old_issues/00235a99
```

Logging and Triage Mechanisms

Results after triage



Logging and Triage Mechanisms

Results after triage

ANDROID BUILDERS SUMMIT

- Majority of issues reproduced in AOSP – reported directly to Google
- 7 issues considered security vulnerabilities, 3 included in Android Security Bulletin from September 2014
- Integer overflows in libstagefright:
  - CVE-2014-7915, CVE-2014-7916, CVE-2014-7917

ANDROID BUILDERS SUMMIT

Agenda

Introduction

Fuzzing Media Content in Android

Data generation

Fuzzing the Stagefright framework

Logging & Triage mechanisms

Fuzzing Stagefright with AFL

Fuzzing Stagefright with AFL

The American Fuzzy Lop fuzzing tool

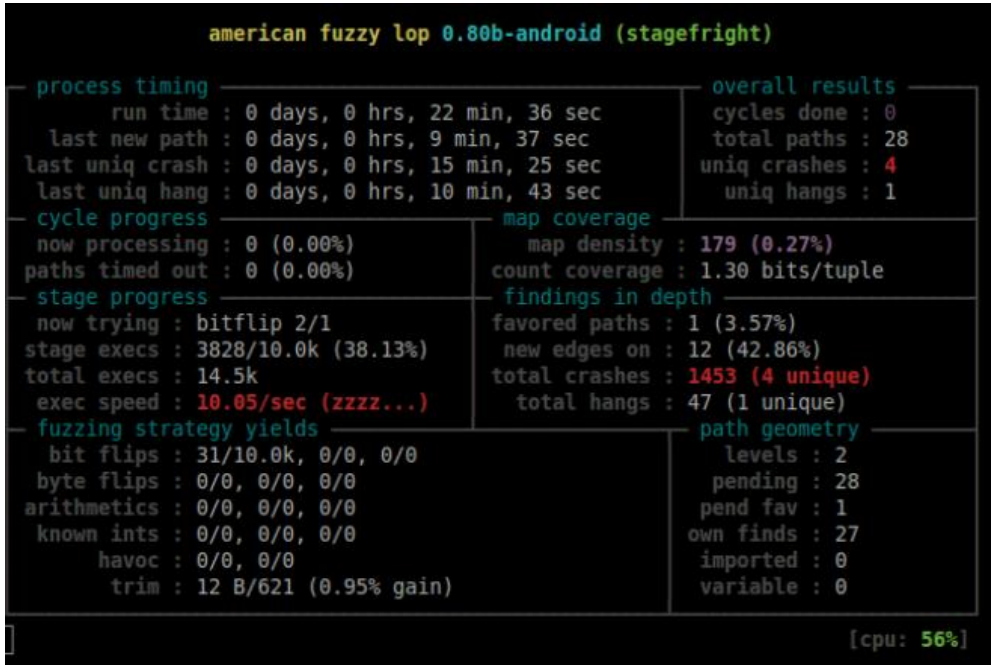
ANDROID BUILDERS SUMMIT

- Instrumentation based fuzzing tool
- Targeted binaries need to be compiled with afl-gcc (wrapper over gcc)
- Two fuzzing modes: dumb-mode, instrumented-mode
- Instrumented mode detects changes to program control flow to find new code paths
- Detects both crashes and hangs and sorts out the unique issues

Fuzzing Stagefright with AFL

AFL on Android

- Build instrumented binary like a regular Android module
- Use environment variables (afl-gcc built as wrapper over gcc toolchain from Android)



ANDROID BUILDERS SUMMIT

https://github.com/fuzzing/MFFA

ioan-alexandru.blanda@intel.com

intel Software

Intel OpenSource TECHNOLOGY CENTER